Optimization as an Environmentally Friendly Action

Silvia Matellán Fernández  
100452093@alumnos.uc3m.es  
Universidad Carlos III de Madrid  
28911 Leganés, Madrid, Spain

Blanca González García  
100454722@alumnos.uc3m.es  
Universidad Carlos III de Madrid  
28911 Leganés, Madrid, Spain

Camino Rodríguez Pérez-Carral  
100445091@alumnos.uc3m.es  
Universidad Carlos III de Madrid  
28911 Leganés, Madrid, Spain

Andrea Iguacel Durán  
100452337@alumnos.uc3m.es  
Universidad Carlos III de Madrid  
28911 Leganés, Madrid, Spain

ABSTRACT
The work described in this paper aims to highlight the significance of optimizing computer programs as a crucial step towards reducing the environmental impact of computational activities. We will use as an example the optimization of a specific program, simulating fluid interactions, to showcase the tangible benefits of such actions.

ACM Reference Format:

1 INTRODUCTION
As the world struggles with the escalating concerns of climate change and environmental sustainability, the role of technology in contributing to a greener future becomes increasingly important.

This work deals with the environmental footprint of computing, emphasizing the energy consumption and carbon emissions associated with running resource-intensive applications. It highlights the urgent need for sustainable computing practices and introduces optimization as a powerful strategy to mitigate these environmental concerns [1].

This work presents a particular case study of a fluid simulation using the Smoothed Particle Hydrodynamics method, which solves the Navier-Stokes equation at time steps. In this method, the fluid is modelled as a collection of particles, which have interactions impacting their acceleration, position and velocity. At each time step, particle acceleration is updated based on nearby particles, and, using these new values, the particle movement is computed as follows:

\[
\begin{align*}
\vec{p}_i &= \vec{p}_i + \vec{h}v_i \cdot \Delta t + \vec{a}_i \cdot (\Delta t)^2 \\
\vec{a}_i &= \vec{a}_i + \vec{a}_i \cdot \Delta t
\end{align*}
\]

where \(\vec{p}_i\) is the position of the particle, \(\vec{v}_i\) is the velocity and \(\vec{h}v_i\) is the impulse vector.

In order to speedup simulation time, long-distance particles were discarded. To achieve this, the 3D space is divided in a grid of blocks and only particles in blocks in the nearby are considered.

The simulation processing involves multiple stages executed for each time step: reposition, forces computation, collision processing, movement of particles and boundaries processing. In the repositioning stage, particle coordinates are checked and adjusted to ensure they are in the correct block of the simulation grid. The forces computation stage involves initializing densities and accelerations, increasing densities, transforming densities, and transferring accelerations between particles. Particles collisions stage considers collisions with box walls, updating vector accelerations based on specified conditions. The particles motion stage updates particle positions and velocities using calculated accelerations. Lastly, the box boundaries interactions stage addresses collisions with the simulation box boundaries, updating particle properties accordingly. The entire process is repeated for each time step, ensuring the accurate simulation of particle dynamics in the given grid.

Several optimizations can enhance the efficiency of the described simulation. Block neighbors storage involves storing information about neighboring blocks to reduce redundant calculations, allowing for more efficient particle repositioning and force computations. Constants extraction optimizes the simulation by extracting constants from repeated computations, reducing redundant calculations and improving overall performance. Compiler optimizations involve leveraging compiler features to enhance code execution, such as loop unrolling or vectorization, which can significantly boost simulation speed. One of our most significant optimizations was the wavefront optimization. It did not only help us improve our program’s execution, but it also solved the problem of a particle being increased twice with the same increment (since \(\vec{a}_{ij} = \vec{a}_{ji}\)). This optimizations works by making each particle only consider those “after” itself, thus reducing the number of computations, and therefore the execution time, by almost half. These optimizations collectively contribute to a more streamlined and faster execution of the simulation, making it more computationally efficient.

This implementation was tested dealing with 4,800 particles and with 15,138 particles. The graphs in figure 1 show both the execution time, and the energy use for both number of particles. As shown in the figures, both execution time and energy use quickly
Figure 1: Execution time (in seconds) and energy use (in Joules) for large and small samples

escalate when dealing with a larger number of particles, making the optimization of the program of crucial importance.

Additionally, program optimization has broader implications. It is worth highlighting the positive cascading effects of adopting optimized code, including enhanced scalability, improved hardware utilization, and extended lifespan of computing infrastructure.

3 CONCLUSIONS AND FURTHER WORK

In conclusion, this poster aims to advocate for the integration of sustainable computing practices by showcasing the environmental benefits derived from the optimization of a fluid simulation program, which are one of the United Nations Sustainable Development Goals [4]. By sharing insights into the process and outcomes of program optimization, it seeks to inspire a broader conversation on the pivotal role that the tech industry can play in contributing to a greener and more sustainable future.

While the described simulation exhibits a series of optimizations such as block neighbors storage, constants extraction, and compiler optimizations, there is further potential for performance improvement through parallelization. Parallelizing the simulation can leverage multi-core processors or specialized hardware like GPUs, distributing the computational load across multiple processing units. By concurrently processing independent tasks, such as particle computations within different blocks or collisions checks, parallelization can significantly reduce the overall simulation time. This approach becomes particularly impactful as hardware architectures increasingly emphasize parallel processing capabilities. Implementing parallelization techniques would not only capitalize on modern computing resources but also unlock the full potential for speeding up the simulation, making it even more adept at handling larger-scale scenarios and real-time simulations.

REFERENCES